Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Boštjan Buh

Sistem obogatene resničnosti za simulacijo bolezni človeškega vida na osnovi Oculus Rift očal

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Peter Peer

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:
Sistem obogatene resničnosti za simulacijo bolezni človeškega vida na osnovi Oculus Rift očal

Oculus Rift očala nadgradite tako, da bodo iz dveh kamer, ki jih spredaj pritrdite na očala, prejemala živi sliki, znotraj očal pa ustvarite realen občutek gledanja skozi očala. Nato preučite bistvene bolezni človeškega vida in implementirajte ustrezen izbor le-teh z uporabo nadgrajenih Oculus Rift očal. Simulacije bolezni morajo delovati v realnem času. Omogočena naj bo sočasna uporaba različnih simulacij. Omogočite tudi, da lahko lastnosti simulacij spreminjamo med samim izvajanjem le-teh. Z enostavnim eksperimentom potrdite (ali ovržite) smiselnost uporabe Oculus Rift očal za simulacijo bolezni vida.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Boštjan Buh, z vpisno številko **63100080**, sem avtor diplomskega dela z naslovom:

Sistem obogatene resničnosti za simulacijo bolezni človeškega vida na osnovi Oculus Rift očal

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Petra Peera,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 15. septembra 2014 Podpis avtorja:

Diploma je nastala pod mentorstvom Petra Peera, ki se mu iskreno zahvaljujem za vse nasvete ter pripombe. Posebna zahvala gre tudi kolegom Jesenki Pibernik, Lidiji Mandić, Jurici Dolić in Bojanu Kanižaju z zagrebške Fakultete za grafiko za njihov prispevek pri snovanju sistema.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
2D	2-Dimensional	Dvo-dimenzionalno
3D	3-Dimensional	Tri-dimenzionalno
AR	Augmented Reality	Obogatena resničnost
DLL	Dynamic Link Library	Dinamična povezovalna knjižnica
GLEW	OpenGL Extension Wrangler	OpenGL razširitveni paket
HD	High Definition	Visoka ločljivost
LCD	Liquid Crystal Display	Zaslon s tekočimi kristali
OpenCV	Open source Computer Vision library	Odprtokodna knjižnica za računalniški vid
OpenGL	Open Graphics Library	Odprta grafična knjižnica
VR	Virtual Reality	Navidezna resničnost

Povzetek

Cilj diplomske naloge je povezava očal Oculus Rift in dveh spletnih kamer s programom za simulacijo bolezni vida. Oculus Rift očala so očala za prikaz navidezne resničnosti, ki se uporabljajo predvsem v igrah. Izvedli smo simulacije petih različnih bolezni z obogateno resničnostjo: barvne slepote, tunelskega vida, retinopatije, dioptrije ter degeneracije makule. Za implementacijo smo uporabljali knjižnico Unity za povezavo očal s kamerama in simulacijo bolezni vida ter Visual Studio aplikacijo za nastavitve posameznih bolezni vida. Na koncu smo kvaliteto simulacij bolezni vida preverili na zagrebški Fakulteti za grafiko.

Ključne besede: bolezni oči, simulacija bolezni, Oculus Rift, Unity, C#, OpenCV.

Abstract

The main goal of this diploma thesis is to link Oculus Rift glasees and two cameras with a program to simulate eye diseases. Oculus Rift glases are glasses for VR and are mostly used in computer gaming. We performed simulation of five different diseases with help of AR: color blindness, tunel vision, retinopathy, myopia/hypermetropia and macular degeneration. For implementation we used Unity library to connect glasses with cameras and to simulate eye diseases, and Visual Studio for control simulation aplication. Finnaly we checked quality of simulation of eye diseases at the Faculty of graphics in Zagreb, Croatia.

Keywords: eye disease, disease simulation, Oculus Rift, Unity, C#, OpenCV.

Poglavje 1

Uvod

Mnogokrat se ob omembi bolezni vida kot so barvna slepota, tunelski vid, dioptrija, itd. pojavi vprašanje, kako osebe s temi boleznimi vida vidijo. Cilj diplomske naloge je implementacija simulacij različnih bolezni vida in prikazati osebam brez bolezni vida kako vidijo osebe s posamezno boleznijo vida. Vse to mora delovati v realnem času s čim manjšo zakasnitvijo. To znanje se lahko potem uporablja pri načrtovanju različnih uporabniških vmesnikov. Za dosego cilja smo uporabili očala za navidezno resničnost Oculus Rift, spletni kameri za stereo prikaz in razvili namensko programsko opremo. Rešitev zajema dve aplikaciji: aplikacijo za zajem slike kamer in simulacijo bolezni ter aplikacijo za upravljanje nastavitev prikaza bolezni. Za zajem slik ter implementacijo posameznih bolezni smo uporabljali knjižnico EmguCV, ki je C# vmesnik za knjižnico OpenCV, je spisana v jeziku C++.

V drugem poglavju smo opisali sestavo očesa ter bolezni barvna slepota, tunelski vid, retinopatija, dioptrija ter degeneracija rumene pege. Nato smo v tretjem poglavju v razvojnem okolju Unity pripravili program, s katerim smo zajeli sliki iz kamer, ki smo jih namestili na Oculus Rift očala, ter ju ustrezno prikazali na zaslonu Oculus Rift očal. V tem poglavju smo tudi opisali implementacijo vseh petih simulacij bolezni vida ter na koncu še optimizirali izvajanje programa. V četrtem poglavju smo opisali ločeno aplikacijo, v kateri lahko med samim izvajanjem simulacije spreminjamo nastavitve posamezne bolezni vida. Peto poglavje opisuje demonstracijo simulacij bolezni vida na zagrebški Fakulteti za grafiko ter naše ugotovitve. V šestem poglavju smo podali zaključke našega dela.

Poglavje 2

Bolezni vida

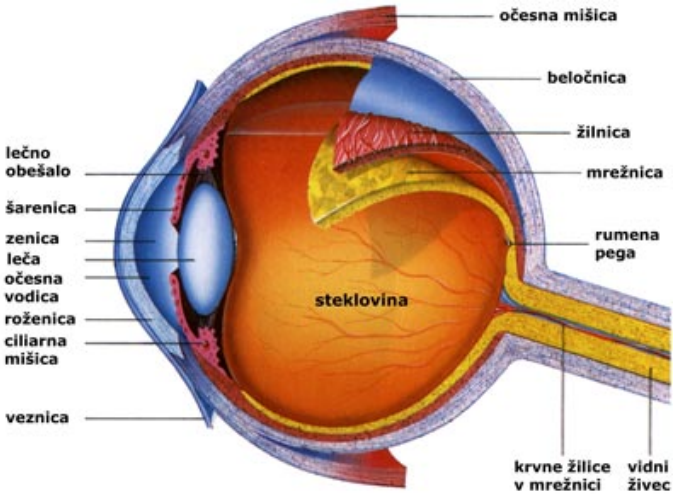
Oči s pomočjo receptorjev (čutilnih celic) na mrežnici na zadnji strani očesnega zrkla zaznavajo svetlobo ter barvo. Svetloba pride do mrežnice tako, da potuje skozi zenice in prekatne tekočine do leče, ta pa lomi svetlobne žarke, ki se nato zberejo na mrežnici (slika 2.1) [1]. Receptorji se delijo na čepke ter paličice. Paličice so občutljive za svetlobo in ne zaznavajo barv. Čepki pa zaznavajo barvo in se delijo na tri vrste glede na barvo, ki jo zaznavajo: rdeče, zelene in modre. Največ čepkov se nahaja na ti. rumeni pegi, ki se nahaja točno nasproti očesne leče. V primeru okvare čepkov, paličic ali leče ločimo več bolezni vida [2].

2.1 Barvna slepota

Barvna slepota je nezmožnost zaznavanja vseh ali le nekaterih barv. Barvna slepota je lahko dedna in se prenaša s kromosomom X ali kromosomom 7 lahko pa se pojavi zaradi poškodbe možganov, živcev in očesa ali izpostavitve kemikalijam. Poznamo delno ali popolno barvno slepoto [3, 4]. Slika 2.2 prikazuje vid barv brez barvne slepote.

Ljudje s popolno barvno slepoto ne ločijo med posameznimi barvami, saj barve zaznavajo le kot belo-sive odtenke. Popolna barvna slepota je zelo redka in se pojavlja le pri 0,00003% moških in žensk. Pojavlja se v dveh oblikah [6]:

- Achromatopsia ali monokromacija paličic, pri kateri čepki ne sprejemajo barve ali pa mrežnica ne vsebuje čepkov. Poleg ne zaznavanja barv je po-



Slika 2.1: Sestava očesa

Slika 2.2: Dojemanje barv brez barvne slepote

Slika 2.3: Dojemanje barv z monokramacijo

slabšano tudi zaznavanje svetlobe (slika 2.3). Achromatopsia je zelo redka.

- Monokromacija čepkov, pri kateri obstajajo tako čepki kot paličice, vendar samo ena vrsta. Te osebe lahko zaznavajo vzorce, ne morejo pa razlikovati posameznih odtenkov. Najbolj pogosta oblika je monokromacija čepkov za zaznavo modre barve, pri kateri manjkajo čepki za zaznavo rdeče in zelene barve.

Delno barvno slepoto v grobem delimo na dve skupini: dikromacija ter anomalna trikromacija. V primeru delne barve slepote se manjkajoča barva nadomesti z odtenki drugih dveh barv. Dikromacija je tip bolezni, pri kateri je v celoti odsotna ena vrsta čepkov. Ta tip bolezni je manj pogost kot anomalna trikromacija in jo ima manj kot 2,5% moških ter 0,03% žensk [6]. Glede na tip manjkajočih čepkov ločimo tri bolezni [4, 5]:

- Protanopija, kjer manjkajo rdeči čepki. Oseba težko razloči med zeleno-rumeno-rdečimi barvami in jih vidijo kot temen odtenek modrozelene barve. Prav tako v primeru rdeče, oranžne, rumene barve slabše ločijo temnejše odtenke od svetlejših. Barve kot sta vijolična ter roza vidijo kot odtenke modre barve. Protanopika se prenaša preko kromosoma X. Primer vida s to barvno slepoto je na sliki 2.4 [7].

- Devtranopija, kjer manjkajo zeleni čepki. Tudi te osebe težko razločijo med zeleno-rumeno-rdečimi barvami, vendar za razliko od protonopije lahko ločijo

Slika 2.4: Dojemanje barv s protanopijo

Slika 2.5: Dojemanje barv z devtranopijo

temnejše odtenke od svetlejših. Devtranopija se prenaša preko kromosoma X. Primer vida s to barvno slepoto je na sliki 2.5 [8].

- Tritanopija, kjer manjkajo modri čepki. Osebe s tritanopijo vidijo modre odtenke kot zatemnjeno zelenkasto barvo, temnejše odtenke modre tudi kot črno. Rumene ne ločijo od roza, vijolično barvo pa vidijo kot rdečo. Tritanopija se prenaša preko kromosoma 7. Primer vida s to barvno slepoto je na sliki 2.6 [9].

Za razliko od dikromacije, kjer manjkajo določeni čepki v celoti, so pri anomalni trikromaciji okvarjeni določeni čepki. Osebe s to vrsto bolezni težje ločijo med svetlimi ter temnimi odtenki. Glede na to, kateri čepki delno manjkajo, ločimo tri bolezni:

Slika 2.6: Dojemanje barv s tritanopijo

- Protanomalija, kjer so podobno kot pri protanopiji okvarjeni čepki rdeče barve, zato osebe težko zaznavajo različne odtenke rdeče, temne odtenke pa zaznajo kot črne. Prisotna je pri 1% moških ter 0,003% žensk [7].

- Devtranomalija, ki ima okvarjene zelene čepke. To je najbolj pogosta oblika barvne slepote in je prisotna pri skoraj 5% moških in 0,4% žensk. Osebe s to boleznijo slabše zaznavajo zeleno barvo in so zato zelenkasti odtenki bolj rdečkasti. V temnem okolju temnozeleno vidijo kot rdečo. Za razliko od oseb s protanomalijo, osebe z devtranomalijo ne izgubijo svetlosti odtenkov [8].

- Tritanomalija, ki ima okvarjene modre čepke. Za razliko od ostalih barvnih slepot se ta prenaša na kromosomu 7 namesto na kromosomu X, zato je enako pogosto pri ženskah in moških. To je najbolj redek tip barvne slepote, ima ga le približno 0,0002% ljudi [9].

2.2 Tunelski vid

Tunelski vid ali izguba perifernega vida je bolezen, ki zmanjša zorni kot vida (slika 2.7). Periferni vid uporabljamo za zaznavanje objektov, ki niso v središču našega pogleda. Glavna razloga za to bolezen sta glavkom ter odstop mrežnice. Pri glavkomu je poškodovan vidni živec na mestu izhoda iz očesa, ki skrbi za prenos slike. Za to bolezen so bolj dojemljivi starejši ljudje, lahko pa se pojavi tudi že v otroštvu, ali pa je bolezen prirojena [10].

Slika 2.7: Različne stopnje tunelskega vida

2.3 Retinopatija

Retinopatija oz. diabetična retinopatija se pojavi zaradi mašenja kapilar v mrežnici. Kapilare se najprej le razširijo, čez čas pa tudi počijo in tako povzročijo pikčaste in lisaste krvavitve na mrežnici. V začetku oseba nima težav z vidom, težave se pojavijo z napredovanjem obolenja, ko se začnejo razraščati nove kapilare in tedaj oseba vidi črne lise (slika 2.8). Ta faza se imenuje prolifererativna faza. Naslednjo fazo, ko se pojavijo krvavitve tudi v steklovini imenujemo hermatovitreus, kar lahko v končni fazi pripelje do slepote. Za retinopatijo zbolijo vsi sladkorni bolniki v približno 25. letih. Zdravila proti retinopatiji ni, največ kar lahko naredimo je strogo uravnavanje krvnega sladkorja, saj tako upočasnimo razvoj retinopatije. V zadnjem času se uveljavlja tudi postopek laserske fotokoagulacije mrežnice, ki začasno zaustavi krvavitve v mrežnici in upočasni razvoj bolezni [11, 12].

2.4 Dioptrija

Ločimo dve vrsti dioptrije: kratkovidnost (myopia) in daljnovidnost (hyperopia). Obe se pojavita zaradi nepravilne oblike leče.

Pri kratkovidnosti se svetlobni žarki zberejo že pred mrežnico in se ob zadetku v mrežnico že razhajajo. Osebe s kratkovidnostjo vidijo dobro na blizu ter slabo na daleč (slika 2.9). Take osebe morajo nositi razpršilno oz. konkavno lečo.

Slika 2.8: Vid osebe z retinopatijo

Slika 2.9: Levo - slika popolnega vida, desno - kratkovidnost

Kratkovidnost se najpogosteje razvije med 8. in 12. letom. Pred in po tem je verjetnost za kratkovidnost zelo majhna. Prisotna je pri približno 25% populacije [13].

Pri daljnovidnosti se svetlobni žarki zberejo šele za mrežnico. Daljnovidne osebe slabo vidijo na blizu (slika 2.10). Osebe z daljnovidnostjo morajo nositi zbiralno oz. konveksno lečo. Prisotna je pri približno 10% populacije [14]. Posebna oblika daljnovidnosti je starovidnost oz. presbiopija. Le ta se pojavlja pri osebah starejših od 40 let, saj s staranjem očesna leča postaja vedno bolj toga, kar deloma onemogoči prilagajanje ukrivljenosti leče [15].

Slika 2.10: Levo - slika popolnega vida, desno - daljnovidnost

2.5 Degeneracija rumene pege

Degeneracija rumene pege oz. degeneracija makule je bolezen, ki prizadene območje rumene pege na mrežnici. Rumena pega vsebuje največ čepkov ter paličic in nam omogoča oster vid. Degeneracija makule se pojavlja predvsem pri starejših ljudeh. Prvi znaki bolezni so popačena slika, namesto ravnih črt vidijo ukrivljene ter manjši kontrast slike (slika 2.11). Bolezen ločimo na dva tipa: suho ter vlažno obliko. Suha oblika je bolj pogosta in milejša oblika. Do izgube vida pride zaradi nalaganja belih ali rumenih depozitov na ali v makulo. Vlažna oblika je manj pogosta in povzroča nenadno in hujšo izgubo vida. Razlog za to je razraščanje krvnih žil in prepuščanje tekočine v središču rumene pege [16].

Poleg omenjenjih bolezni vida obstaja še veliko ostalih kot so npr. motnje v steklovini, siva mrena, ambliopija, odstop mrežnice ipd.

Slika 2.11: Napredovanje retinopatije

Poglavje 3

Implementacija simulacij bolezni vida

Izmed vseh zgoraj opisanih omenjenih smo izbrali naslednjih 5 za implementacijo:

- barvna slepota,
- tunelski vid,
- retinopatija,
- dioptrija,
- degenaracija rumene pege.

Te bolezni vida smo izbrali na podlagi predloga kolegov iz zagrebške Fakultete za grafiko, ki se ukvarjajo s tem področjem.

3.1 Priprava okolja

Oculus Rift so očala, ki so namenjena potopitvi v virtualno resničnost. So prva nizkocenovna očala te vrste na tržišču. Zanimanje zanje se je pokazalo že z njihovim projektom na Kickstarterju, kjer so od zahtevanih 250.000 dolarjev zbrali skoraj 2,5 milijona dolarjev. Oculus Rift je v osnovi sestavljen iz dveh malih LCD zaslonov ter leč, tako da pričara pravo 3D izkušnjo. Vsebuje tudi giroskop, merilec pospeškov ter magnetometer, kar omogoča zaznavanje obračanja v vse smeri. Trenutno obstajata

Slika 3.1: Oculus Rift očala

dve različici. Starejša različica ima ločljivost 640*800 slikovnih elementov na oko in ima zaradi tega bolj nerazločno pikčasto sliko, novejša različica pa vsebuje ločljivost 960*1080 na oko ter ima dodano tudi ločeno kamero, ki se uporablja za zaznavanje premikov očal v prostoru (približevanje, oddaljevanje). V našem projektu smo uporabljali starejšo različico (slika 3.1). Cena obeh različic se giblje okoli 300 dolarjev [18] [19].

Oculus Rift kot vhodno sliko sprejme sliko, ki je sestavljena iz dveh slik, ki sta ena ob drugi ter vsaka zaseda točno polovico slike. Sliki morata biti zamaknjeni, saj le tako dobimo ob gledanju pravi 3D občutek. Oculus nato vzame levo polovico slike za levo oko ter desno polovico za desno oko.

Pred začetkom implementacije smo se odločali med dvema različnima kamerama (slika 3.2), ki sta se že uporabljali za implementacijo skupaj z Oculus Rift očali za navidezno resničnost:

- **Sony PlayStation 3 (PS3) Eye** kamera omogoča največjo ločljivost snemanja 640*480 slikovnih elementov. Prednost kamere je ta, da je zelo majhna, ko se odstrani iz ohišja in je cenejša, slabost pa je njena slaba povezljivost z računalnikom, saj je kamera prvotno namenjena PlayStation konzolam in so potrebni dodatni gonilniki pri povezovanju z računalnikom [20].

- **Logitech HD Webcam C310** kamera omogoča snemanje v ločljivosti 1280*960 slikovnih elementov. Prednost kamere je vsekakor njena ločljivost ter enostavna povezljivost z računalnikom, slabost pa njena cena in njena velikost, ko se jo razstavi [21].

Odločili smo se za Logitech HD Webcam C310 kamero, saj bomo le tako lahko

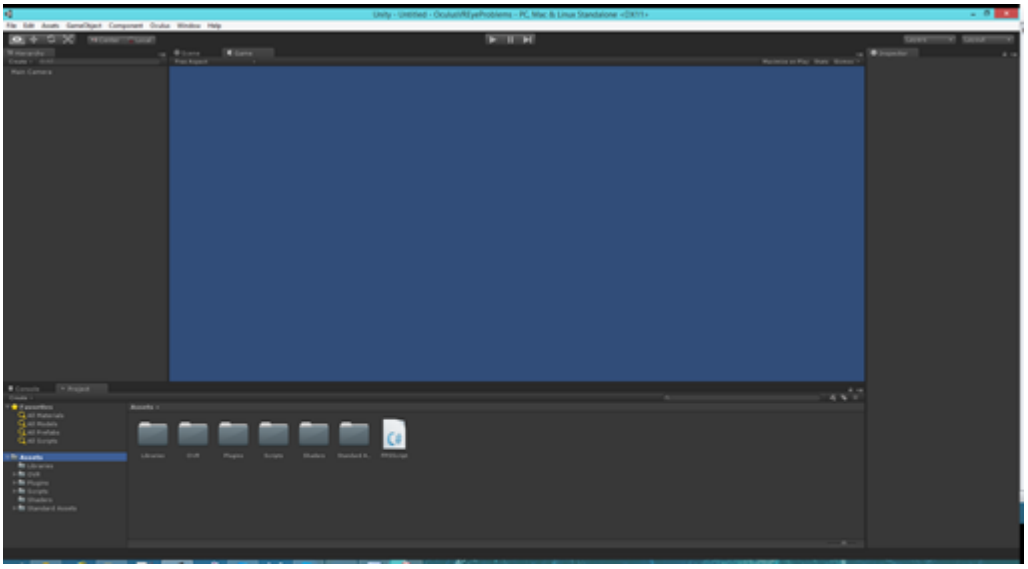
Slika 3.2: Kamera Logitech C310 (levo) ter PS3 Eye (desno)

Slika 3.3: Nova 2,1 mm leča [22]

popolnoma izkoristili ločljivost Oculus Rift očal. S 3D tiskalnikom smo naredili tudi ustrezna nosilca za kameri. Ker pa je vidni kot kamere samo 60°, človeškega očesa pa približno 170°, smo se odločili za dodatno lečo 2,1 mm (slika 3.3) ter tako dosegli vidni kot približno 110°. Pri zajemu slike smo uporabili ločljivost 800*600 slikovnih točk.

Najprej smo nato pripravili projekt, s katerim smo prenesli sliko iz kamer v Oculus Rift očala. Izbirali smo med 2 orodjema:

- **Unity**, ki vsebuje zmogljivo knjižnico ter razvojno okolje. Uporablja se predvsem za izdelavo iger. Unity ima tudi podporo za razvoj za Oculus Rift očala. Razvoj znotraj Unity je razdeljen na 2 dela. Prvi je priprava objektov s pomočjo grafičnega vmesnika, drugi pa je implementacija kode za zahtevnejše stvari s pomočjo programa MonoDeveloper ter programskega jezika C# [24].
- **Microsoft Visual Studio**, ki je splošno okolje za programiranje. Podpira delo z Microsoft .NET platformo, ki predstavlja ogrodje za vse .NET orien-



Slika 3.4: Razvojno okolje Unity

tirana programska orodja in aplikacije. Visual Studio vsebuje pester nabor jezikov, omogoča kreiranje projektov, sestavljenih iz posameznih modulov, ki so lahko sestavljeni iz različnih programskih jezikov. Omogoča tudi hiter in enostaven razvoj ter izvajanje tako enostavnih kot tudi zahtevnejših projektov. Razvoj je omogočen v večih jezikih kot so C#, C++, C, VisualBasic ipd. [23].

Za uporabo smo si najprej izbrali orodje Visual Studio ter programski jezik C#. Za to smo se odločili predvsem zato, ker sta nam Visual Studio okolje ter programski jezik C# že zelo poznana. Za predlogo smo uporabil že narejeno C# aplikacijo, ki je s pomočjo OpenCV ter glew knjižnic že zajemala sliko iz ene kamere. Aplikacijo smo nadgradili tako, da je sprejela 2 kameri ter sliki nastavila eno ob drugi. Dobili smo sliko, ki je bila potrebna za prikaz na Oculus očalih, ampak je program deloval zelo počasi, saj je v najboljšem primeru dopustil le 10 slik na sekundo. Zato smo se odločili za implementacijo z uporabo Unity, saj je Unity bolj optimiziran za delo z Oculus Riftom. Slika 3.4 prikazuje osnovno okno razvojnega okolja Unity.

Po namestitvi Unity ter uvozu dodatka za delo z Oculusom, se nam v orodni vrstici pokaže možnost Oculus. Preko tega lahko poženemo aplikacijo ter dodajamo elemente, ki so namenjeni samo za razvoj za Oculus Rift. Preko tega menija smo

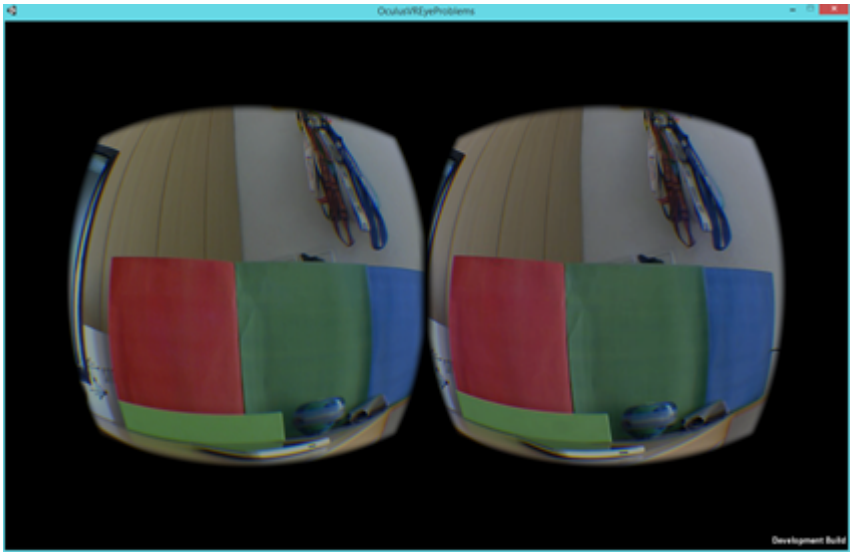
Slika 3.5: Priprava Unity objektov za prikaz slik

dodali nov objekt, ki se imenuje `OVRPlayerController`. `OVRPlayerController` skrbi za pravilen prikaz slike 3D objektov iz igre na Oculusu. V našem primeru smo kot objekte igre uporabili Plane objekt, na katerem bomo prikazali sliko kamer za vsako oko posebej ter za vsako stran dodali še svojo kamero (slika 3.5).

```
public class WebCamTextureRight : MonoBehaviour {
        WebCamTexture webcamTexture;
        // Use this for initialization
        void Start () {
                webcamTexture = new WebCamTexture("Logitech HD Webcam
                    C310", 800, 600, 30);
                renderer.material.mainTexture = webcamTexture;
                webcamTexture.Play();
        }
        // Update is called once per frame
        void Update () {
                renderer.material.mainTexture = webcamTexture;
        }
}
```

Izpis 3.1: Začetna vsebina skripte

Unity omogoča, da za teksture objektov (sliko, ki se prikaže na objektih) uporabimo poljubne skripte, ki so napisane v jeziku C#. Tako smo za vsakega izmed Plane objektov ustvarili skripto, v kateri smo implementirali zajem slik iz kamere ter simulacijo bolezni vida. Vsaka izmed teh skript ima takoj na začetku že pripravljeni dve metodi (izpis 3.1) [24]:



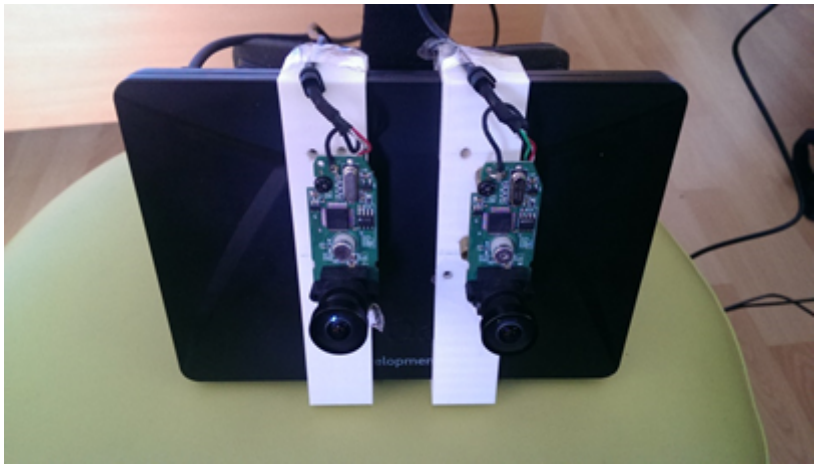
Slika 3.6: Zajem slike iz kamer

- Start, kjer pripravimo vse potrebne spremenljivke ter podatke pred začetkom igre. Izvede se samo na začetku igre.

- Update, ki se izvede po Start metodi in se neprestano izvaja do konca igre. Ta metoda skrbi za posodabljanje slike ter lastnosti objekta.

S pomočje že privzetih knjižnic ter metod, ki jih Unity podpira, smo implementirali enostaven zajem slike ter prikaz na Plane objektu za posamezno oko. Tako smo na zelo enostaven in hiter način brez posebnih dodatnih knjižnic implementirali funkcionalnost, ki smo jo najprej naredili v Visual Studiu (slika 3.6).

Ko smo imeli pripravljen zajem slike iz kamer ter posredovanje le-te v Oculus Rift očala, smo potrebovali še nosilca, s katerima bomo namestili kameri na sprednjo stran očal. Na osnovi modela nosilca smo ga natisnili s 3D tiskalnikom. 3D tiskanje je postopek izdelave trdih objektov iz računalniških 3D modelov s pomočjo aditivnega postopka, pri katerem se v zaporednih plasteh odlaga material. Ta tehnologija se uporablja predvsem za izdelavo prototipov ter v porazdeljeni proizvodnji [25]. Slika 3.7 prikazuje končna Oculus Rift očala z nameščenima kamerama.



Slika 3.7: Oculus Rift očala z nameščenima kamerama

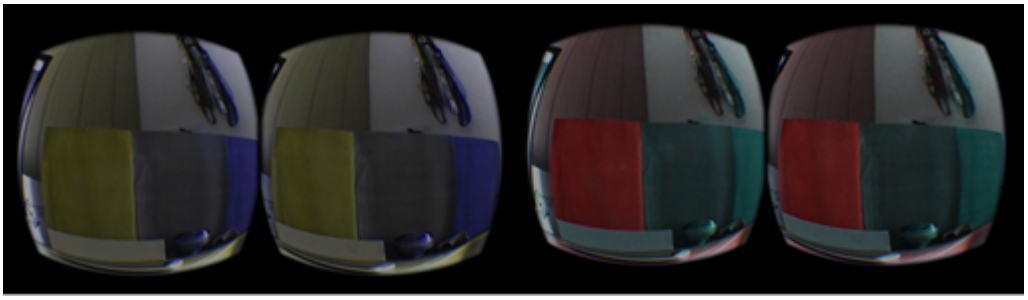
3.2 Barvna slepota

Ko smo uspešno prikazali sliko iz kamer, smo se lotili implementacije simulacij bolezni vida. Najprej smo implementirali bolezen barvne slepote. Osnovna ideja je, da se po slikovnih elementih slike sprehodimo čez sliko ter vsakemu elementu nastavimo ustrezno barvo ter na ta način simuliramo različne barvne slepote. Pripravili smo si nov razred imenovan Integration.cs, v katerem bomo implementirali simulacije bolezni vida ter skrbeli za posodabljanje nastavitev, ki se bodo nastavljale v ločeni aplikaciji. Prav tako smo si na ta način zagotovili, da bomo imeli implementacijo simulacij bolezni vida na enem mestu in se tako znebili nepotrebne podvojene kode.

Pri določeni barvni slepoti se manjkajoča barva nadomesti s preostalima dvema barvama. Na spletu [26] smo našli nekaj osnovnih preslikav barv za simulacijo osnovnih barvnih slepot (slika 3.2). Podatke smo normalizirali, tako da je vsota vrednosti znotraj posamezne barve znašala točno 1 ter si rezultate shranili v seznam matrik velikost 3x3. Nato smo dobljeno sliko iz kamere pretvorili v slikovne elemente ter poklicali metodo `ColorBlindness`. V objektu `Transformation` smo že pred tem shranili matriko barvne slepote, ki jo trenutno simuliramo.

V tej metodi se sprehodimo čez vse slikovne elemente, ki so tipa Color. Objekti tipa Color imajo definirano barvo preko treh glavnih lastnosti:

- R, ki nam pove količino rdeče komponente barve.



Slika 3.8: Simulacija protonopije (levo) ter tritanopije (desno). Slika 3.6 prikazuje vid brez simulacije barvne slepote

- G, ki nam pove količino zelene komponente barve.
- B, ki nam pove količino modre komponente barve.

```
{"Normal", new float[3,3]{/*R:*/{1,0,0}, /*G:*/{ 0,1,0},
    /*B:*/{0,0,1}}},
{"Protanopia", new float[3,3]{/*R:*/{0.567f,0.433f,0},
    /*G:*/{0.558f,0.442f,0}, /*B:*/{0,0.242f,0.758f}}},
{"Protanomaly", new float[3,3]{/*R:*/{0.817f,0.183f,0},
    /*G:*/{0.333f,0.667f,0}, /*B:*/{0,0.125f,0.875f}}},
{"Deuteranopia", new float[3,3]{/*R:*/{0.625f,0.375f,0},
    /*G:*/{0.7f,0.3f,0}, /*B:*/{0,0.3f,0.7f}}},
{"Deuteranomaly", new float[3,3]{/*R:*/{0.8f,0.2f,0},
    /*G:*/{0.258f,0.742f,0}, /*B:*/{0,0.142f,0.858f}}},
{"Tritanopia", new float[3,3]{/*R:*/{0.95f,0.05f,0},
    /*G:*/{0,0.433f,0.567f}, /*B:*/{0,0.475f,0.525f}}},
{"Tritanomaly", new float[3,3]{/*R:*/{0.967f,0.033f,0},
    /*G:*/{0,0.733f,0.267f}, /*B:*/{0,0.183f,0.817f}}}
```

Izpis 3.2: Preslikave barv pri različnih barvnih slepotah

Nato smo nastavili novo vrednost posamezne komponente barve tako, da smo jo sestavili iz vseh treh barv glede na transformacijsko matriko (izpis 3.3).

Slika 3.8 prikazuje rezultat simulacije za protonopijo ter tritanopijo.

Ker smo pri implementaciji dioptrije začeli uporabljati knjižnico OpenCV oz. njen vmesnik EmguCV, smo morali spremeniti večji del programa za simulacijo barvne slepote (in drugih bolezni), algoritem je seveda ostal enak. Prav tako smo

to priložnost uporabili za optimizacijo ter pospešitev programa. Vsi ti postopki so opisani v poglavju 3.7.

```
public  Color[] ColorBlindnes(Color[] pixels)
{
        for (int i = 0; i< pixels.Length; i++ ) {
                Color pixel = pixels[i];
                float r = ((pixel.r * transformation[0, 0]) + (pixel.g
                    * transformation[0, 1]) + (pixel.b *
                    transformation[0, 2]));
                float g = ((pixel.r * transformation[1, 0]) + (pixel.g
                    * transformation[1, 1]) + (pixel.b *
                    transformation[1, 2]));
                float b = ((pixel.r * transformation[2, 0]) + (pixel.g
                    * transformation[2, 1]) + (pixel.b *
                    transformation[2, 2]));
                pixel.r = r;
                pixel.g = g;
                pixel.b = b;
                pixels[i] = pixel;
        }
        return pixels;
}
```

Izpis 3.3: Implementacija simulacije barvne slepote

3.3 Tunelski vid

Tunelski vid je izguba perifernega vida, ki nam omogoča večji zorni kot gledanja. Simulacijo smo implementirali na način, da smo na začetku igre oz. pri zamenjavi ločljivosti zajema slike v aplikaciji za upravljanje simulacije izračunali razdaljo od središča slike za vsako slikovno točko. To smo izračunali ločeno za levo ter desno sliko, saj smo le tako lahko zagotovili ustrezno sliko na obeh očesih.

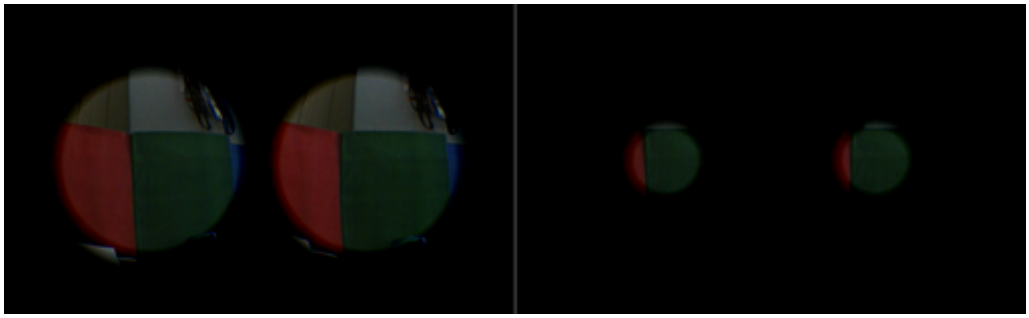
```
public  Color[] TunelVision(Color[] pixels)
{
        for (int i = 0; i< pixels.Length; i++ ) {
                if (radiouses[i] > TunelRadious)
                {
                        float trans = (float)transparence/100;
                        double difference = (radiouses[i] -
                            TunelRadious) * gradient;
                        float r = Math.Max(0, pixels[i].r -
                            (float)difference) ;
                        float g = Math.Max(0, pixels[i].g -
                            (float)difference);
                        float b = Math.Max(0, pixels[i].b -
                            (float)difference);

                        // Apply transparency
                        pixels[i].r = ((pixels[i].r - r)*trans)+r;
                        pixels[i].g = ((pixels[i].g - g)*trans)+g;
                        pixels[i].b = ((pixels[i].b - b)*trans)+b;
                }
        }
        return pixels;
}
```

Izpis 3.4: Implementacija simulacije tunelskega vida

Nato smo si pripravili še tri spremenljivke, ki se bodo nastavljale preko aplikacije za upravljanje simulacije:

- Polmer, ki nam pove polmer velikosti vidnega polja v slikovnih točkah.
- Gradient, ki nam pove kako, hiter je prehod iz vidnega polja v nevidno polje.
- Prozornost, ki nam pove, kolikšna je še prepustnost slike na ne vidnem polju (v %).

Tunelski vid smo nato implementirali tako, da smo za vsako točko preverili, če je polmer v trenutni točki večji kot polmer iz nastavitev, nato smo nastavili čim bolj črno barvo ter na koncu aplicirali še transparentnost (izpis 3.4).



Slika 3.9: Simulacija tunelskega vida. Slika 3.6 prikazuje vid brez simulacije tunelskega vida

Slika 3.9 prikazuje dve različni simulaciji tunelskega vida.

3.4 Retinopatija

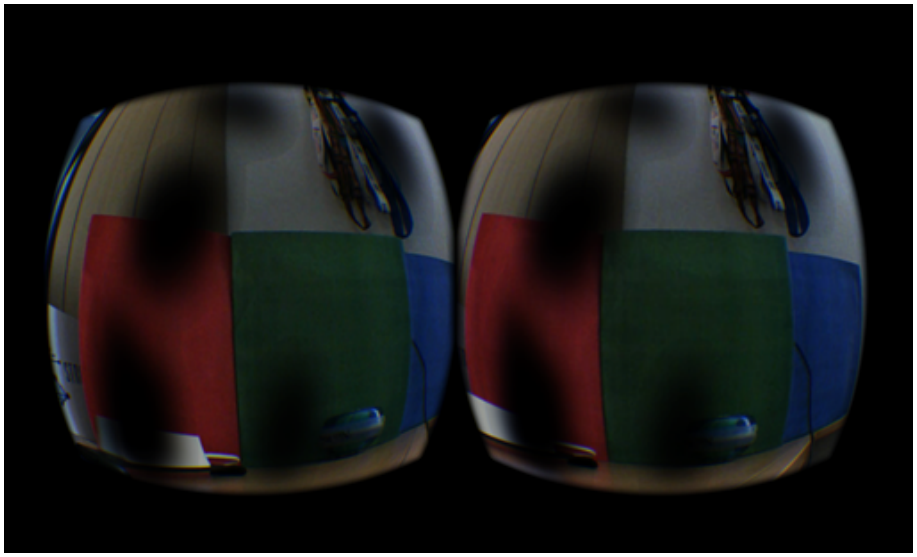
Pri retinopatiji se osebi prikazujejo črne pege oz. pike, ki lahko malo prepuščajo svetlobo ali pa popolnoma zabrišejo del slike ter so lahko večje ali manjše ter različnih oblik. Zaradi vseh teh različnih sprememb smo retinopatijo implementirali na način, da smo uporabili predlogo, ki je črno-bela slika, ki prikazuje, kje je okvara vida ter v kakšni meri.

Takoj pri zagonu oziroma pri spremembi nastavitev, se v pomnilnik shrani slika, ki se uporablja kot predloga retinopatije (spremenljivka `retinapthyImage`). Nato se sprehodimo čez osnovno sliko ter primerjamo slikovne točke na istih mestih. Če slikovna točka v predlogi ni bela, potem trenutno slikovno točko potemnimo za toliko kot je temna slikovna točka predloge (izpis 3.5).

Slika 3.10 prikazuje primer simulacije retinopatije.

3.5 Dioptrija

Za implementacijo dioptrije smo rabili podatek o globini, saj smo le na ta način lahko zameglili oddaljene/bližnje predmete za implementacijo dioptrije. Zaradi tega smo za zajem slik ter delo s slikami uporabili knjižnico OpenCV oz. njen vmesnik EmguCV [28], kar je pripeljalo do popolne spremembe implementacije ostalih simulacij. Spremembe so opisane v poglavju 3.7.



Slika 3.10: Primer retinopatije. Slika 3.6 prikazuje vid brez simulacije retinopatije

```
public  Color[] ProcessRetinopathy(Color[] pixels)
{
        for (int i = 0; i< pixels.Length; i++ ) {
                Color retinopathy = retinapthyImage[i];
                if (retinopathy.r < 0.9960784)
                {
                        float value = Math.Max(1.1f * retinopathy.r -
                            0.1f, 0);
                        pixels[i].r = Math.Max(0, pixels[i].r -
                            (pixels[i].r *(1-value)));
                        pixels[i].g = Math.Max(0, pixels[i].g -
                            (pixels[i].g *(1-value)));
                        pixels[i].b = Math.Max(0, pixels[i].b -
                            (pixels[i].b *(1-value)));
                }
        }
        return pixels;
}
```

Izpis 3.5: Implementacija simulacije retinopatije

Za uspešno pridobivanje podatka o globini ter simulacijo dioptrije so bili potrebni naslednji postopki:

- Zajem kalibracijskih slik.
- Kalibracija kamer, s čimer smo zagotovili, da sta obe kameri vzporedni. S tem je bil objekt na obeh slikah na enaki višini.
- Izračun globinske slike iz kalibriranih kamer.
- Simulacija dioptrije.

Zajem slik ter kalibracija se izvajata v aplikaciji za upravljanje simulacije, izračun globine ter simulacija dioptrije pa v Oculus Rift aplikaciji.

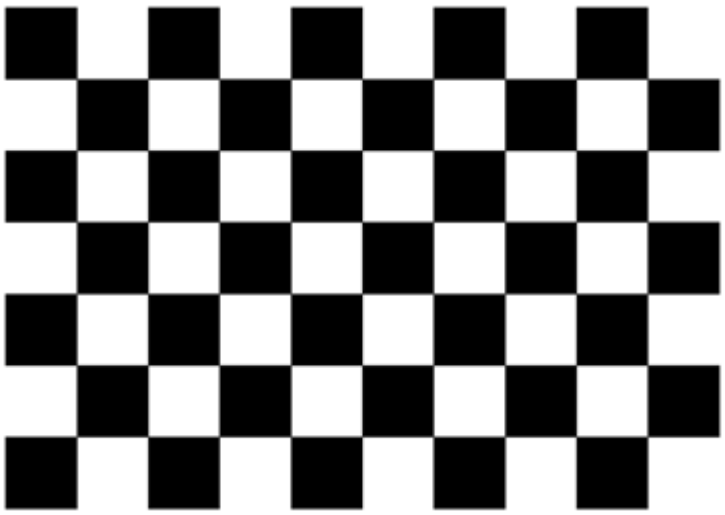
3.5.1 Zajem slik

Za kalibracijo kamer je bilo potrebno istočasno zajeti sliko iz obeh kamer. Na obeh slikah je morala biti vidna slika šahovnice (slika 3.11), ki smo jo natisnili na papir velikost A4. Med samim zajemanjem smo morali šahovnico premikati, tako da je bila zajeta iz različnih zornih kotov ter razdalj. Za preverjanje, ali je šahovnica res vidna na sliki, smo uporabili že privzete metode EmguCV, ki na podani sliki poišče vse kote na šahovnici. Dobljene rezultate iskanja kotov šahovnice smo si ločeno zapomnili za levo ter desno kamero, saj smo jih uporabili v naslednjem koraku [33].

3.5.2 Kalibracija kamer

Pri zajemanju slik se privzeto zajema 50 slik, a smo zaradi večje natančnosti izračuna zajemali 100 slik. Ko smo uspešno zajeli vseh 100 slik šahovnice, smo morali poskrbeti, da sta obe kameri vzporedni. Zopet smo uporabili metodi EmguCV knjižnice [29, 30, 31]:

- Metodo `CameraCalibration.StereoCalibrate`, ki iz zajetih kotov za posamezno kamero izračuna potrebni matriki za premik ter vrtenje.
- Metodo `CvInvoke.cvStereoRectify`, ki iz prejšnjih rezultatov izračuna perspektivno transformacijsko matriko, ki se uporablja za izračun globine.



Slika 3.11: Šahovnica za kalibracijo kamer pri zajemanju globinske slike

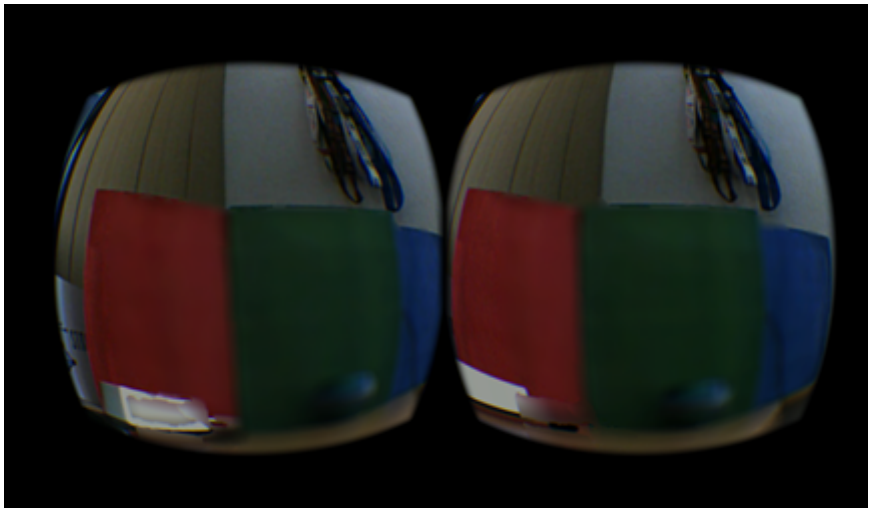
Ker sta ti dve metodi zelo potratni (formiranje perspektivne transformacijske matrike se izvaja približno 15 minut), si perspektivno transformacijsko matriko shranimo [33].

3.5.3 Izračun globinske slike

Ko imamo pripravljeno matriko za izračun globine, lahko izračunamo potrebno globinsko sliko. Zopet uporabimo že obstoječo metodo EmguCV `StereoSGBM.FindStereoCorrespondence`, ki sprejme sivinsko sliko iz leve ter desne kamere ter perspektivno transformacijsko matriko ter nam vrne sivinsko sliko, ki prikazuje globinsko sliko. Na globinski sliki so elementi, ki so blizu, bele barve, elementi v ozadju pa bolj črne barve [33].

3.5.4 Simulacija dioptrije

Sedaj ko imamo pripravljeno globinsko sliko, lahko implementiramo tudi dioptrijo. Za zajeto sliko iz kamere si pripravimo več slik z različno intenzivno zameglitvijo slike z metodo, tako da večkrat kličemo metodo `Image.SmoothBlur`, ki sliko zamegli, z različno intenzivnostjo in si slike zapomnimo. Nato glede na izbrano vrsto dioptrije, ki je lahko kratkovidnost ali daljnovidnost ter podatkom o globini na posamezni slikovni točki, uporabimo slikovno točko iz ustrezno zamegljene slike.



Slika 3.12: Simulacija daljnovidnosti, zgornji del slike je dobro viden. Izhodiščno ter globinsko sliko prikazuje slika 4.6

Na ta način se nam lahko zameglijo le oddaljeni oz. le bližnji predmeti [34].

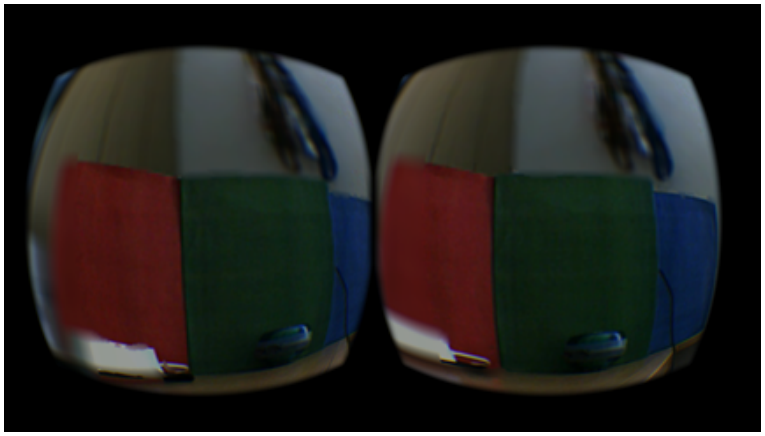
Poleg izbire dioptrije smo pri implementaciji upoštevali še naslednje nastavitve:

- Intenzivnost, ki nam pove največjo stopnjo zamegljenosti oddaljenega/bližnjega predmeta.
- Razdaljo, s katero lahko kontroliramo pri kateri razdalji se začne/konča dioptrija.

Slika 3.12 prikazuje simulacijo daljnovidnosti ter slika 3.13 prikazuje simulacijo kratkovidnosti.

3.6 Degeneracija rumene pege

Pri degeneraciji rumene pege oseba v začetnem stadiju vidi le valovito sliko, kasneje pa se mu na tem mestu pojavi še črna pega. Tako smo morali pri implementaciji degeneracije rumene pege najprej poškodovano območje narediti valovito ter nato še zatemniti območje. Odločili smo se za podoben pristop kot pri retinopatiji (poglavje 3.4), kjer smo uporabili sliko za predlogo, saj se lahko tudi degeneracija rumene pege razlikuje od osebe do osebe, pripravljeno pa smo imeli tudi že



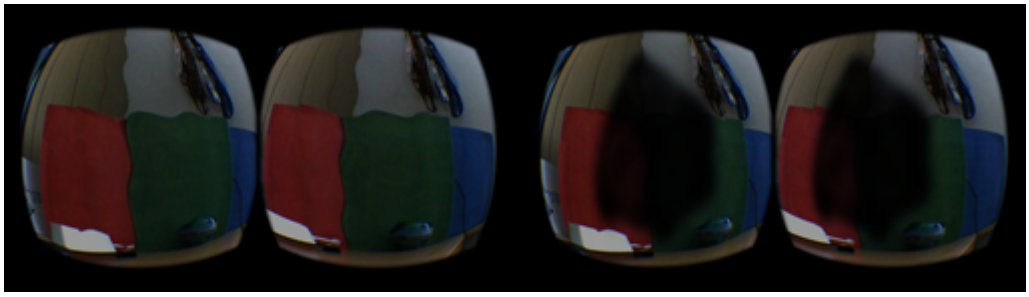
Slika 3.13: Simulacija kratkovidnosti, spodnji del slike je dobro viden. Izhodiščno ter globinsko sliko prikazuje slika 4.6

zatemnitev poškodovanega območja. Predloga je sestavljena enako kot pri retinopatiji, le da jo tukaj uporabljamo tudi za preslikavo valovite slike na poškodovano območje.

```
float dist(float a, float b){
        return sqrt(a * a + b * b);
}
float4 frag(v2f_img i) : COLOR
{
        float4 posit = 0;
        float f = sin(dist(i.uv, i.uv.y)*_WaveSize)
                  + sin(dist(i.uv.x, i.uv.y)*_WaveSize)
                  + sin(dist(i.uv.x, i.uv.y)*_WaveSize);
        i.uv.x = i.uv.x+((f/_WaveSize)/_WavingRation);
        i.uv.y = i.uv.y+((f/_WaveSize)/_WavingRation);
        posit= tex2D( _MainTex , i.uv.xy);
        return posit;
}
```

Izpis 3.6: Senčilnik za valovito sliko

Za pripravo valovitega območja smo se odločili za uporabo senčilnikov (angl. shaders). Senčilniki so programi, ki se uporabljajo predvsem za prikaz svetlobe ter senc pri obdelavi slike, lahko pa se uporabljajo tudi za posebne efekte. V našem



Slika 3.14: Simulacija različnih stopenj degeneracije rumene pege. Slika 3.6 prikazuje vid brez simulacije degeneracije rumene pege

primeru smo s pomočjo senčilnikov implementirali valovito sliko tako, da smo s pomočjo sinusne funkcije zamikali posamezne slikovne točke slike. Za vsako točko se glede na njeno pozicijo ter velikost vala iz nastavitev izračuna nova pozicija (izpis 3.6). Tako smo celotno sliko spremenili v valovito sliko ter si jo shranili za kasnejšo uporabo.

Ko smo imeli pripravljeno valovito sliko ter predlogo oblike bolezni in zatemnitve, smo na vsakem mestu, kjer v predlogi ni bila bela barva, obstoječo slikovno točko zamenjali s slikovno točko iz valovite slike. Nato smo vpeljali še spremenljivko intenzivnosti, ki v odstotkih pove v kolikšni meri je potrebno zatemniti sliko, kjer v predlogi ni bela barva (slika 3.14).

3.7 Optimizacija delovanja

Pri implementaciji dioptrije smo morali vpeljati knjižico EmguCV ter zaradi tega preurediti aplikacijo. Tako smo se odločili, da izkoristimo to priložnost za optimizacijo delovanja, predvsem zaradi dodatne možnosti za istočasno prikazovanje različnih bolezni vida. Tako smo se optimizacije lotili v dveh korakih.

3.7.1 Dodana zunanja zanka

Najprej smo pripravili zunanjo zanko, v kateri smo se sprehodili čez vse slikovne točke slike. Nato smo popravili metode barvne slepote, tunelskega vida ter retinopatije tako, da je naredila operacijo samo nad eno slikovno točko. Kot smo omenili smo to optimizacijo izvajali ob implementaciji dioptrije, ko še nismo imeli

implementirane simulacije degeneracije rumene pege. Pri barvni slepoti je bila to edina sprememba.

Pri tunelskem vidu smo spremenili vrednost spremenljivk gradient ter radij tako, da vsebujeta vrednost v odstotkih. Radij se nato aplicira kot odstotek največjega polmera, gradient pa kot odstotek nevidnega polja (izpis 3.7).

```
static private  void TunelVision (ref byte red, ref byte green, ref
    byte blue, int column, int row)
{
        double radious = radiouses [column, row];
        double recalculatedRadious = maxRadious * (TunelRadious / 100);
        if (radious > recalculatedRadious) {
                float trans = (float)transparence / 100;
                double difference = radious - recalculatedRadious;

                double gradientToPoint =
                    ((maxRadious-recalculatedRadious)*gradient/100);

                double percentageGradient = difference /
                    gradientToPoint;

                byte redTemp = (byte)(Math.Max (0, red -
                    (float)(percentageGradient * 255)));
                byte greenTemp = (byte)(Math.Max (0, green -
                    (float)(percentageGradient * 255)));
                byte blueTemp = (byte)(Math.Max (0, blue -
                    (float)(percentageGradient * 255)));

                // Apply transparency
                red = (byte)(((red - redTemp) * trans) + redTemp);
                green = (byte)(((green - greenTemp) * trans) +
                    greenTemp);
                blue = (byte)(((blue - blueTemp) * trans) + blueTemp);
        }
}
```

Izpis 3.7: Optimizirana implementacija simulacije tunelskega vida

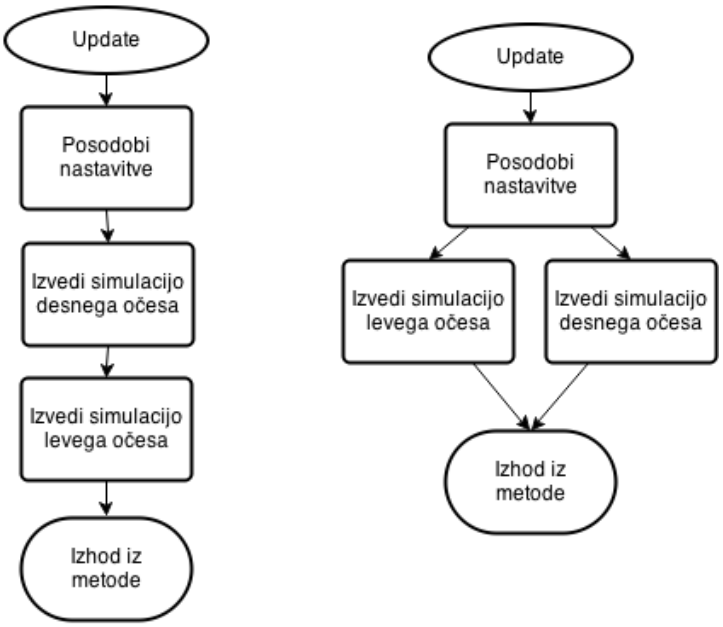
Pri retinopatiji smo dodali dva nova parametra. In sicer intenzivnost ter zamik. Intenzivnost nam pove v kakšni meri se prikazuje črna pika, z zamikom pa zamikamo simulacijo retinopatije desnega očesa tako, da sta sliki poravnani. Zamik smo implementirali zato, ker smo imeli probleme pri opazovanju zelo bližnjih ali pa zelo oddaljenih stvari, saj je vidni razmik drugačen (izpis 3.8).

```
static private  void ProcessRetinopathy (ref byte red, ref byte green,
    ref byte blue, int column, int row, BitmapData bmpData, bool isLeft)
{
        if (!isLeft) {
                column += retinapthyIndent;
                if (column >= bmpData.Width) {
                                column = bmpData.Width - 1;
                }
        }

        int posIndex = (bmpData.Height - 1 - row) * bmpData.Width +
            column;
        UnityEngine.Color changeColor = retinapthyImageB [posIndex];
        if (changeColor.r < 0.9960784) {
                float percentageBlack = (1 - changeColor.r) *
                    (float)retinapthyIntensity / 100;
                // apply intensity
                red = (byte)Math.Ceiling (red - (red *
                    percentageBlack));
                blue = (byte)Math.Ceiling (blue - (blue *
                    percentageBlack));
                green = (byte)Math.Ceiling (green - (green *
                    percentageBlack));
        }
}
```

Izpis 3.8: Optimizirana implementacija simulacije retinopatije

3.7.2 Večnitnost

Nit v programu je najmanjše samostojno zaporedje programskih ukazov, ki uporablja naslovni prostor v pomnilniku od nadrejenega procesa [35]. En proces lahko



Slika 3.15: Izvajanje programa pred in po implementaciji večnitnosti

vsebuje več niti, ki se izvajajo ločeno na procesorju. Ker Unity v osnovi poganja glavno zanko v eni niti, smo se odločili za vpeljavo dveh niti. Vsaka izmed niti izvaja simulacijo bolezni za eno oko. Večnitno izvajanje smo implementirali znotraj Update metode, ki se izvaja vsak obhod igre (slika 3.15).

Z dodajanjem nove zanke se čas izvajanja v primeru ene simulacije ni zmanjšal, se je pa zmanjšal v primeru simulacije večih bolezni vida hkrati. Tako se je pred tem pri istočasni simulaciji treh bolezni vida sprehodil trikrat čez celotno sliko, sedaj pa to naredi le enkrat.

Z večnitnostjo smo dosegli pohitritev tako pri simulacije ene bolezni vida kot pri istočasni simulaciji različnih bolezni. Aplikacija se izvaja skoraj enkrat hitreje.

Poglavje 4

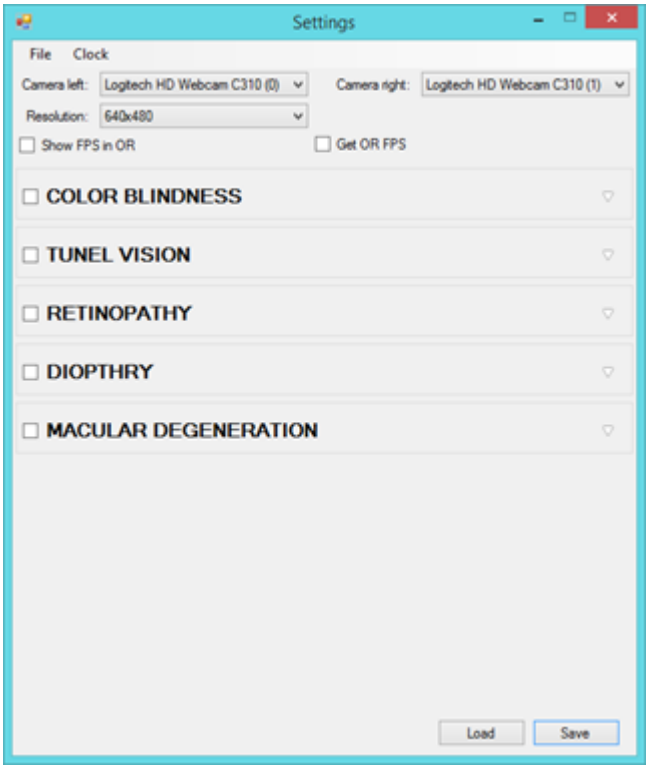
Aplikacija za upravljanje simulacije

Ker naša Oculus Rift aplikacija vsebuje zelo veliko različnih parametrov, smo si naredili tudi aplikacijo za upravljanje simulacije (slika 4.1). Z njo lahko nastavljamo vse parametre bolezni med samim izvajanjem ter izbiramo med različnimi boleznimi vida. Bolezni vida lahko med sabo tudi kombiniramo (npr. barvna slepota s tunelskim vidom). Aplikacija je s simulacijo povezana tako, da nastavitve shrani na disk, potem pa simulacija prebere spremembe v primeru, da je bila datoteka spremenjena od zadnjega branja ter osveži prikaz. Ta postopek pa lahko predstavlja ozko grlo simulacije, saj mora aplikacija s simulacijo vsakič dostopati do diska ter prebrati datoteko. Ta problem bi lahko rešili s pomočjo medprocesne komunikacije [34].

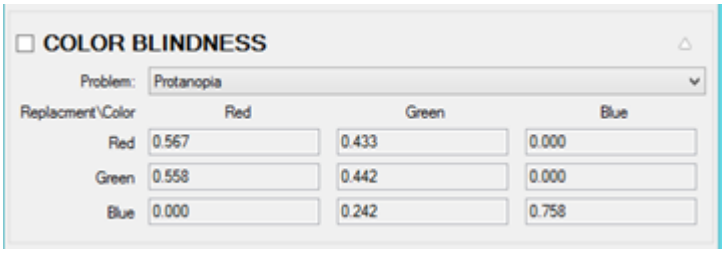
Pri nastavitvah barvne slepote (slika 4.2) lahko izbiramo med več vnaprej določenimi barvnimi slepotami, lahko pa tudi sami vnesemo vrednosti za preslikavo barv. To nam omogoča simulacijo (več kot) vseh različnih barvnih slepot.

Za tunelski vid smo pripravili naslednje parametre (slika 4.3):

- Polmer v odstotkih glede na najdaljšo razdaljo od sredine.
- Gradient nam pove razdaljo spreminjanja slike iz vidne slike v črno. Vrednost je v odstotkih glede na razdaljo ne vidnega polja.
- Prozornost nam pove, kolikšna je še prepustnost slike na ne vidnem polju.



Slika 4.1: Aplikacija za upravljanje simulacije



Slika 4.2: Parametri barvne slepote

Slika 4.3: Parametri tunelskega vida

Slika 4.4: Parametri retinopatije

Vrednost je v odstotkih.

Nastavitve retinopatije (slika 4.4) so bile naslednje:

- Slika predloge, ki je črno-bela slika in prikazuje popačenje slike.
- Intenzivnost, s katero lahko nastavljamo intenzivnost črnih peg. Vrednost je v odstotkih.
- Zamik, ki se uporablja za poravnavo desne slike simulacije retinopatije z levo sliko simulacije retinopatije. Vrednost je v odstotkih.

Pri nastavitvah za dioptrijo (slika 4.5) imamo na voljo naslednje parametre:

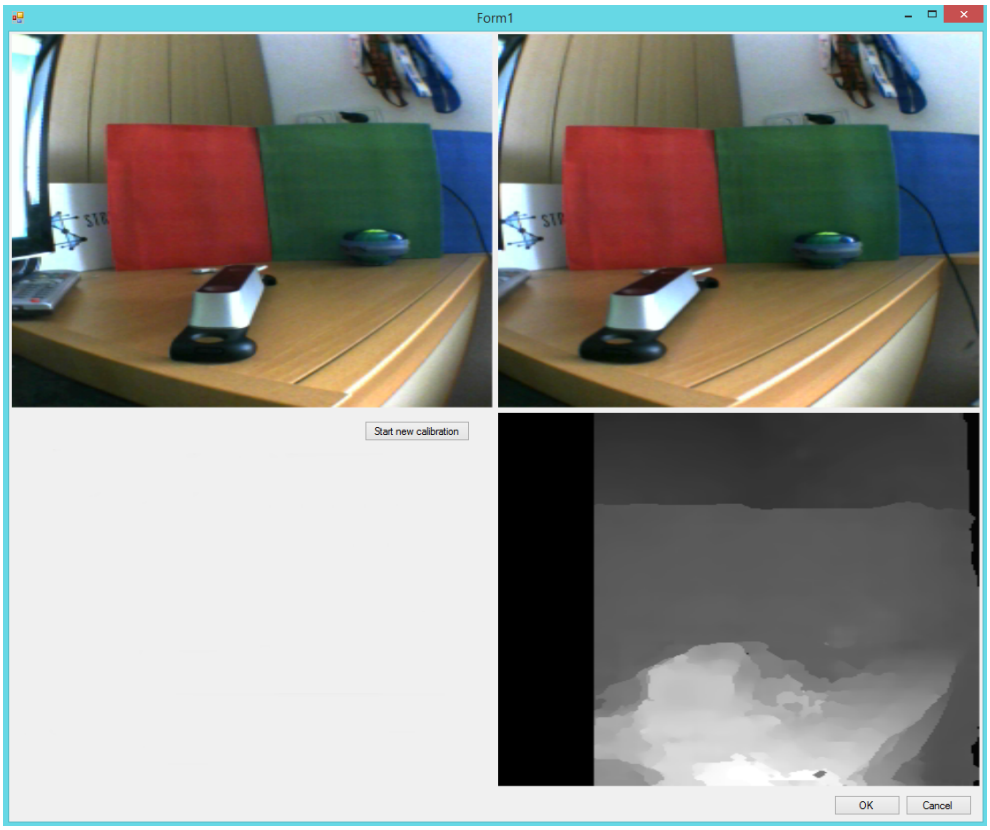
- Tip dioptrije, kjer lahko izbiramo med kratkovidnostjo ter daljnovidnostjo.
- Intenzivnost nam pove največjo stopnjo zamegljenosti oddaljenega/bližnjega predmeta. Večja intenzivnost odraža bolj zamegljeno sliko. Izbirno območje je med 1 in 25.
- Razdaljo, s katero lahko kontroliramo pri kateri razdalji se začne/konča dioptrija. Vrednost je v odstotkih.

Slika 4.5: Parametri dioptrije

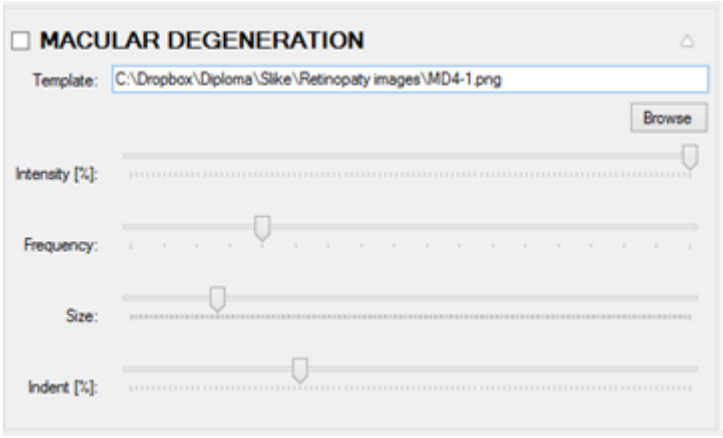
Na voljo imamo tudi kalibracijo kamer (slika 4.6), kjer v postopku kalibracije zajamemo slike šahovnice ter nato dobimo perspektivno transformacijsko matriko.

Nastavitve degenarcije rumene pege (slika 4.7) so podobne nastavitvam retinopatije, z dodatnim parametri za valovito slike. Na voljo imamo tako parametre:

- Slika predloge, ki je črno-bela slika in prikazuje popačenje slike.
- Intenzivnost, s katero lahko nastavljamo intenzivnost črnih peg. Vrednost je v odstotkih.
- Frekvenca, s katero nastavimo frekvenco sinusnih valov. Večja frekvenca odraža večjo gostoto valov. Izbirno območje je med 1 in 20.
- Velikost, s katero nastavimo velikost sinusnih valov v slikovnih točkah. Izbirno območje je med 1 in 300.
- Zamik, ki se uporablja za poravnavo desne slike simulacije z levo sliko simulacije. Vrednost je v odstotkih.



Slika 4.6: Okno za kalibracijo kamer pri dioptriji



Slika 4.7: Parametri degeneracije rumene pege

Poglavje 5

Kvalitativna ocena razvitega sistema

Preliminarno oceno našega sistema smo dobili, ko smo se dogovorili za sestanek s kolegi z zagrebške Fakultete za grafiko, ko so prišli na simpozij v Slovenijo. Bili so tako navdušeni nad simulacijami bolezni vida, da so nas povabili, da demonstriramo naš sistem v Zagrebu.

Tako smo kvalitativno ocenitev razvitega sistema izvedli z demonstracijo na zagrebški Fakulteti za grafiko, v Laboratoriju za ocenjevanje uporabniške izkušnje. Na demonstracijo so povabili kolege iz Fakultete za elektrotehniko in računalništvo, Pedagoške fakultete ter Fakulteta za grafiko Univerze v Zagrebu ter predstavnike nekaterih podjetij. Demonstracije se je udeležilo 20 oseb.

Vsaki osebi smo prikazali simulacije bolezni vida po vnaprej pripravljenem scenariju (sliki 5.1 in 5.2):

- Barvni slepoti protanopijo ter tritanopijo.
- Tunelski vid.
- Retinopatijo.
- Za zabavo smo demonstrirali tudi spremembo dojemanja barv po meri, z zamenjavo modrega in rdečega barvnega kanala smo namreč ljudje postali modrokožči in tako bili podobni avatarjem oz. smrkcom.

Slika 5.1: Oseba med simulacijo bolezni vida

Slika 5.2: Oseba med simulacijo retinopatije

Simulacije so bile zelo dobro sprejete. Skoraj vsi uporabniki so dali pozitiven odziv. Zanimiv je bil odziv kolegice, ki se raziskovalno ukvarja z boleznimi vida: „Zdaj res končno vem, kako se počutijo in vidijo ti ljudje“. Čeprav je uporabnike sprva zmotila slabša ločljivost Oculus Rift očal, eno izmed oseb je obšla tudi manjša slabost ob uporabi, so ostali komentarji potrjevali smiselnost nadaljnega razvoja in uporabe takšnega sistema. Zadovoljni so bili nad odzivom simulacije, saj skoraj ni bilo zaznati zakasnitve. Med samo simulacijo so imeli občutek realnosti, kot da je takšen svet v resnici pred njimi, kot da imajo posamezno bolezen vida. Cilj popolne potopitve v takšen navidezni oziroma obogaten svet je tako bil dosežen.

Zapisana ocena sistema je še toliko zanesljivejša, ker se veliko oseb, ki so se udeležili demonstracije, raziskovalno ukvarja z boleznimi vida in poznajo ostala simulacijska orodja za te bolezni.

Kvalitativni eksperiment je tako potrdil, da je naša simulacija z očali Oculus Rift uporabna za simulacijo bolezni vida.

Za še boljšo simulacijo bolezni je potrebno nadgraditi očala na novejšo različico 2, saj imajo slednja veliko boljšo ločljivost.

Poglavje 6

Zaključek

Zadani cilj smo uspešno implementirali: pripravili smo aplikacijo, ki nam v realnem času simulira pet različnih bolezni vida ter omogoča tudi mešanje med njimi. Problem, ki še vedno ostaja, je ločljivost Oculus Rift očal. Potrebna je tudi dodatna optimizacija algoritmov, kar pa ni tako pereč problem.

Z uporabo razvitega sistema smo ugotovili, da Oculus Rift očala lahko uporabljamo pri simulaciji bolezni vida, vendar z določenimi omejitvami. Glavna omejitev je seveda ločljivost slike očal. Naslednja omejitev je strojna, saj je pri simulaciji visoke kvalitete slike potrebna velika procesorska ter grafična moč za hitro simulacijo bolezni.

V nadaljevanju bo sledila še primerjava razvitega sistema z Zimmerman paketom, ki se prav tako uporablja za simulacijo različnih bolezni vida.

Aplikacija ima še veliko točk za možno izboljšavo ter popravke:

- Dodamo lahko še veliko bolezni.
- Dodamo ločene simulacije za posamezno oko.
- Lahko ga nadgradimo tako, da bi bila kompatibilna z novo verzijo Oculus Rift očal ter s tem zagotovimo boljšo uporabniško izkušnjo pri uporabi.
- Dodatno optimiziramo simulacije tako, da ne bo potrebna tako velika procesorska ter grafična moč za izvajanje simulacije.
- Lahko ga pretvorimo v aplikacijo za Android ter iOS platformo in tako omogočimo uporabo na telefonih s pomočjo pripomočkov kot je npr. Google

Slika 6.1: Google Cardboad VR očala

Cardboard VR očala (slika 6.1).

- Lahko ga uporabimo za obogatitev resničnosti v kakšnem drugem primeru simulacije ipd.

Slike

Izpisi kode

Literatura

[1] (2014) Očesno zrklo – Zgradba očesa – Optika Rene Pirc. Dostopno na: http://www.optika-pirc.com/?menu_item=sl_zgradba

[2] (2014) Kako zaznavamo barve. Dostopno na: http://projekti.gimvic.org/2003/2a/timko_barvila/TIMKO_Anja/html/kakoZazBarva.htm

[3] (2014) Barvna slepota – Očesne napake – Optika Rene Pirc. Dostopno na: http://www.optika-pirc.com/?menu_item=sl_barvnaSlepota

[4] D. Flück, *Color blind essentials*, spletna knjiga, Švica, 2014. Dostopno na http://www.color-blindness.com/wp-content/documents/Color-Blind-Essentials.pdf

[5] B. Wong, Points of view – Color blindness, *Nature methods*, 8(6):441–450, 2011

[6] (2014) Monochromacy – Complete Color Blindness. Dostopno na: http://www.color-blindness.com/2007/07/20/monochromacy-complete-color-blindness/

[7] (2014) Protanopia – Red-Green Color Blindness | Colblindor. Dostopno na: http://www.color-blindness.com/protanopia-red-green-color-blindness/

[8] (2014) Deuteranopia – Red-Green Color Blindness | Colblindor. Dostopno na: http://www.color-blindness.com/deuteranopia-red-green-color-blindness/

[9] (2014) Tritanopia – Blue-Yellow Color Blindness | Colblindor. Dostopno na: http://www.color-blindness.com/tritanopia-blue-yellow-color-blindness/

[10] (2014) Peripheral Vision Loss – Tunnel Vision Causes and Treatments. Dostopno na: http://www.allaboutvision.com/conditions/peripheral-vision.htm

[11] (2014) Diabetic retinopathy – Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/wiki/Diabetic_retinopathy

[12] (2014) Diabetična retinopatija – Očesne bolezni – Optika Rene Pirc. Dostopno na: http://www.optika-pirc.com/?menu_item=sl_retinopatija

[13] (2014) Myopia (Nearsightedness) Definition, Causes and Treatment. Dostopno na: http://www.allaboutvision.com/conditions/myopia.htm

[14] (2014) Hyperopia (Farsightedness) – AllAboutVision.com. Dostopno na: http://www.allaboutvision.com/conditions/hyperopia.htm

[15] (2014) Starovidnost ali presbiopija. Dostopno na: http://www.optikas.com/starovidnost-ali-presbiopija.html

[16] (2014) Macular degeneration – Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/wiki/Macular_degeneration

[17] (2014) Vitreous Opacities – Flashes and Floaters Medical Information. Dostopno na: http://www.medicineonline.com/articles/v/2/Vitreous-Opacities/Flashes-and-Floaters.html

[18] (2014) Oculus Rift: Step Into the Game by Oculus – Kickstarter. Dostopno na: https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game

[19] (2014) Rift – Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/wiki/Oculus_Rift

[20] (2014) PlayStation Eye – Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/wiki/PlayStation_Eye

[21] (2014) Logitech HD Webcam C310 Technical Specifications – Logitech FAQ. Dostopno na: http://logitech-en-amr.custhelp.com/app/answers/detail/a_id/17181/%03/logitech-hd-webcam-c310-technical-specifications

[22] (2014) 2.1mm 160-Degree Wide Angle Lens for Security Cameras and Webcams – Free Shipping – DealExtreme. Dostopno na: http://eud.dx.com/product/2-1mm-160-degree-wide-angle-lens-for-security-cameras-and-webcams-844015237#.VBNNcluUc40

[23] S. Uranič, *Visual C# .NET*, Pasadena, Slovenija, 2010. Dostopno na: http://uranic.tsckr.si/VISUAL%20C%23/VISUAL%20C%23.pdf

[24] (2014) Unity – Manual: Unity Manual. Dostopno na: http://docs.unity3d.com/Manual/

[25] (2014) 3D-tiskanje – Wikipedija, prosta enciklopedija. Dostopno na: http://sl.wikipedia.org/wiki/3D-tiskanje

[26] (2014) Canvas + ColorMatrix = Color Blindnes. Dostopno na: http://web.archive.org/web/20081014161121/http://www.colorjack.com/labs/colormatrix/

[27] (2014) 3D Imaging with NI LabVIEW – National Instruments. Dostopno na: http://www.ni.com/white-paper/14103/en/

[28] S. Shi, *Emgu CV Essentials*, Packt publishing, Velika Britanija, 2013

[29] G. Bradski, A. Kaehler, *Learning OpenCV*, O'Reilly Media, ZDA, 2008

[30] R. Laganière, *OpenCV 2 Computer Vision Application Programming Cookbook*, Packt Publishing, VB, 2011

[31] P. Peer, Gradnja globinskih panoramskih slik s postopkom mozaičenja, *magistrska naloga*, Fakulteta za računalništvo in informatiko Univerze v Ljubljani, Slovenija, 2001

[32] (2014) Interprocess Communications (Windows). Dostopno na: http://msdn.microsoft.com/en-us/library/windows/desktop/aa365574(v=vs.85).aspx

[33] (2014) Stereo Imaging – Emgu CV: OpenCV in .NET (C#, VB, C++ and more). Dostopno na:
http://www.emgu.com/wiki/index.php/Stereo_Imaging

[34] B. A. Barsky, Vision-Realistic Rendering: Simulation of the Scanned Foveal Image from Wavefront Data of Human Subjects, *ACM International Conference Proceedings Series*, zbornik 73, str. 74–87, New York, ACM Press, 2004

[35] (2014) Slovar informatike. Dostopno na:
http://www.islovar.org/izpisclanka.asp?id=4427&oznaci=1